

S-Plus *Short Course*

– Part 2 –

These notes are mostly based on

- The S-PLUS manuals — available online at MathSoft's webpage, www.mathsoft.com, for downloads (as PDF-files). Also available on the Stat Dept Computer System (at `/opt/local/splus5.1/doc`, see the files `unixug.pdf`, `statman1.pdf` and `statman2.pdf`).
- Venables and Ripley's book on S-PLUS: *Modern Applied Statistics with S-plus*, Springer-Verlag. See www.stats.ox.ac.uk/pub/MASS3.

The Schedule

This second part of this S-PLUS Short Course will go through:

Graphics and EDA Exploratory Data Analysis through graphics in S-PLUS. Two datasets will be used; the toxicity data created in part 1 plus the dataset `fuel.frame`, already in S-PLUS.

Classical Univariate Tests t-test, χ^2 -test and other classical univariate tests.

Statistical Models Use the toxicity data to fit models already discussed in class, print and plot the results, and do basic diagnostics.

All commands used in this part are in the text file available on the course webpage:

www.stat.ohio-state.edu/~sses/Stat693C-Sp2000

The Datasets

We will use more than one dataset to explore the graphical and model-fitting capabilities of S-PLUS.

The `tox` data

Recall the `tox` data created in part 1:

```
> tox
      d  n  r      p
1 0.10 47  8 17.02128
2 0.15 53 14 26.41509
3 0.20 55 24 43.63636
4 0.30 52 32 61.53846
5 0.50 46 38 82.60870
6 0.70 54 50 92.59259
7 0.95 52 50 96.15385
```

This `data.frame` was created by `read.table` — if for some reason it can't be found, enter:

```
> tox <- read.table("/home/gardar/pub/toxicity.dat",
+                  header=T)
> names(tox) <- c('d','n','r') ## change names
> tox$p <- tox$r/tox$n * 100   ## percentage killed
```

The fuel.frame data

This `data.frame` comes with **S-PLUS** (and is used in their manuals). The data contain 5 variables that measure characteristics of 60 automobiles:

```
> dim(fuel.frame)
[1] 60  5
> summary(fuel.frame)
      Weight      Disp.      Mileage
Min.:1845      Min.: 73.0      Min.:18.00
1st Qu.:2571    1st Qu.:113.8      1st Qu.:21.00
Median:2885      Median:144.5      Median:23.00
Mean:2901      Mean:152.1      Mean:24.58
3rd Qu.:3231    3rd Qu.:180.0      3rd Qu.:27.00
Max.:3855      Max.:305.0      Max.:37.00

      Fuel      Type
Min.:2.703    Compact:15
1st Qu.:3.704    Large: 3
Median:4.348    Medium:13
Mean:4.210      Small:13
3rd Qu.:4.762    Sporty: 9
Max.:5.556      Van: 7
>
```

where `Disp.` is the displacement of the engine.

On attaching data.frames

Everything in S-PLUS is an *object*, both what you create and what is there already (functions and data). The objects are stored at different locations and the command `search` gives the way S-PLUS *searches* for objects that you want to use. For an example, in this S-PLUS session:

```
> search() ## where S-Plus looks for objects
[1] ".Data" "splus" "stat" "data"
[5] "trellis" "main"
>
```

That is, first S-PLUS searches `.Data` (your working directory where your objects that you create are stored), then the `splus` location, and so on.

By default, the function `ls` lists the objects stored in the *first* location (position 1) given by `search`:

```
> args(ls) ## the arguments to ls(), and def.
function(pattern, pos = 1)
>
```

The command `ls(pos=5)` would list all objects in `trellis`; location number 5 in the search list.

It will often be convenient to *attach* `data.frames` to the search list so specific columns can be referred *directly* by

name instead of needing to specify both the name of the `data.frame` and the column (i.e., instead of `tox$r` just do `r` at the prompt):

```
> attach(tox) ## attaching the tox data.frame
> search()
[1] ".Data"      "tox"        "splus"      "stat"
[5] "data"       "trellis"   "main"
> ls(pos=2)
[1] "d" "n" "p" "r"
> detach(2) ## then detach the data.frame
NULL
> search()
[1] ".Data"      "splus"      "stat"      "data"
[5] "trellis"   "main"
>
```

Warning: If there is an object in `.Data` that has the same name as one of the columns in `tox`, then **S-PLUS** will find that object first instead of using the column in `tox`.

Graphics

Much of exploratory data analysis (EDA) involves graphing (visualizing) the data.

There are two types of graphics available for **S-PLUS** for **UNIX**

Traditional Graphics These are the original tools for plotting in **S-PLUS**.

Trellis Graphics library This is a library of functions (using the traditional plotting functions) focusing on EDA in multivariate data.

We will first learn how to use some of the traditional plotting methods and then move to the Trellis library.

Traditional Graphics

Plots are plotted on a *graphic device*. To start a graphic device in UNIX, type either `motif()` or `trellis.device()`. In more recent versions of S-PLUS, the graphic device is opened *automatically*, if not open already, when a plotting function is executed.

The plot Function and Graph Arguments

The most basic plot function is `plot`, which creates *scatter* plots. My first plot:

```
> attach(tox)
> plot(d,p)
>
```

A scatter plot of percentage killed (`p`) versus concentration (`d`) is made.

Various additional *arguments* can be given to the `plot` function

Labels To label axes and give a title (and sub-title), enter:

```
> plot(d,p,main="Toxicity Data",ylab="% killed",
+      xlab="Concentration",sub="(what ever!)")
>
```

Type The argument `type` specifies what to draw (points, lines — or both). By default, `type='p'` for *points*, other options include `'l'` for *lines*, `'b'` for both points and lines and `'n'` for *nothing* — see help for `plot.default`:

```
> plot(d,p,type="b")
>
```

Lines and Points The argument `lty` (= some integer) gives *line type* (solid, dashed, etc.). `lwd` (= some integer) gives the *line width*. `pch` (= any number between 1 and 18 or some character) gives the *plotting character*. `cex` (= a real number) gives the *size* of characters/symbols relative to `x`. `col` (= integer) gives color.

```
> plot(d,p,type="b",lty=3,lwd=3,pch=16,cex=2)
>
```

— problem: applies to everything plotted.

— solution: build plots step-by-step by *adding* to the *graphic device*.

For further information about the `plot` function and its arguments, see the help on `plot.default`, `title` and `par`.

The *lines* and *points* Functions — and Relatives

We have used the `plot` function to create graphs in just one command. Now, we are going to build 'nice' graphs in a few steps.

First, update the `tox` data by adding two columns: log-transformation of `d` and logit-transformation of `p`:

```
> detach('tox')
> tox$log.d <- log(tox$d)
> tox$logit.p <- log(tox$p/(100-tox$p))
> tox
```

	d	n	r	p	log.d	logit.p
1	0.10	47	8	17.02128	-2.30258509	-1.5841201
2	0.15	53	14	26.41509	-1.89711998	-1.0245043
3	0.20	55	24	43.63636	-1.60943791	-0.2559334
4	0.30	52	32	61.53846	-1.20397280	0.4700036
5	0.50	46	38	82.60870	-0.69314718	1.5581446
6	0.70	54	50	92.59259	-0.35667494	2.5257286
7	0.95	52	50	96.15385	-0.05129329	3.2188758

```
> attach(tox)
>
```

Let's make a fancy plot using the transformed variables:

```
> par(mar=c(5,5,6,5)) ## change the margins
> plot(log.d,logit.p, type='n', ## plot nothing
+       axes=F, ## no axes
```

```
+      xlab='',ylab='')          ## no labels
> box()  ## make a box, def. is bty='o', try 'l'
> axis(1)  ## add axis to side 1, bottom
> axis(2)  ## and to side 2, left
> lines(log.d,logit.p,type='b',pch='      ',cex=3)
> axis(4,at=logit.p,  ## put ticks at logit.p
+      labels=round(p,1),  ## the labels used
+      srt=90,  ## rotate the labels
+      tck=-par()$tck,  ## ticks inside
+      cex=0.8,  ## reduce the character size
+      mgp=c(2,0.5,0))  ## loc of (Title,Lab,Axs)
> axis(3,at=log.d,labels=d,
+      tck=-par()$tck,cex=0.8,mgp=c(2,0.5,0))
> title(xlab='log concentration',
+      ylab='logit of % killed')
> mtext('The toxicity data',  ## margin text
+      adj=0,  ## adjust to left (1 is right)
+      side=3,line=4,cex=1.5)
> mtext('% killed',side=4,line=2.5,cex=0.8)
> mtext('concentration',side=3,line=2.5,cex=0.8)
> title(sub=date(),cex=0.5,adj=1)  ## current date
> abline(h=0,lty=2)  ## horizontal line at 50%
> legend(locator(1),  ## use mouse to pick loc.
+      '50% killed',bty='n')  ## bty='n', no box
>
```

In addition to the `lines` function used above, there are `points` for points and `text` for text; for example:

```
> plot(log.d,logit.p,type='n')
```

```
> text(log.d,logit.p,paste(round(p), '%', sep=' '))  
>
```

What is used as symbols are the percentages ('17%', '26%', etc.) created by the `paste` functions, which 'pastes' % to the percentage numbers (`p`) rounded to 0 (the default for `round`) digits.

Detach the `tox` data

```
> detach('tox')  
NULL  
>
```

Other Plotting Functions

There are number of plotting functions in S-PLUS. We name a few:

boxplot Produces side by side boxplots.

qqplot,qqnorm and qqline The function `qqplot` compares the distribution of two samples via a Q-Q plot, `qqnorm` compares one sample with the *normal* distribution, and `qqline` adds a fitted straight line to the plot.

hist Plots histogram for a sample.

Two examples using the `fuel.frame` data are:

```
> attach(fuel.frame)
> boxplot(split(Mileage,Type),varwidth=T)
> qqnorm(Mileage)
> qqline(Mileage)
> hist(Mileage)
> detach('fuel.frame')
NULL
>
```

See the help for these functions for more detailed examples and use of arguments.

Trellis Graphics

The Trellis library is very handy for EDA of multivariate data.

The `xyplot` function

The `xyplot` creates *scatter* plots. In its simplest form:

```
> xyplot(logit.p ~ log.d, data=tox)
>
```

There are two arguments given, a `formula` and `data`. The formula specifies what to plot, in this case `logit.p` versus `log.d`, and the `data` argument, a `data.frame` object, says where to find the variables.

Other arguments of `xyplot`, and in general of Trellis plotting functions, are `xlab` and `ylab`, `main` and `sub` (as in the `plot` function). To control how and what to plot, the arguments `type`, `pch`, `lty`, `lwd` and `cex` can all be given.

For more information, see the help for `xyplot`, where a detailed example is given, and `trellis.args`, where various options for arguments are given.

Conditioning and Other Plotting Functions

The real power of Trellis plotting functions is obvious in the following examples:

```
> xyplot(Mileage ~ Weight | Type, data=fuel.frame)
>
```

What is plotted is `Mileage` versus `(~) Weight` *conditional* (`|`) on `Type`; that is, there are 6 types of automobiles and therefore 6 *panels* created on the graphic device, and a scatter plot made within each panel for one type of automobile.

A few other Trellis functions are:

```
> bwplot(Type ~ Mileage, data=fuel.frame) ## boxplot
> histogram(~ Mileage | Type, data=fuel.frame)
> qqmath(~ Mileage | Type, data=fuel.frame)
>
```

Again, see the help files for more detailed examples using the functions above.

Printing and Exporting Graphs

To print the graph shown on the *graphic device*, select

`Print` under `Graph` in the graphic device window — don't try this now (it will print on the printer in CH-341).

It is easy to export a graph to a *postscript* file (which can be converted to *pdf* file, if needed). Example:

```
> trellis.device(postscript,file='myplot.ps',
+               horizontal=T)
> xyplot(Mileage ~ Weight | Type, data=fuel.frame)
> dev.off()
```

Generated postscript file "myplot.ps".

```
motif
```

```
  2
```

```
>
```

The file `myplot.ps` is then created in the same folder where `S-PLUS` was started.

Those without access to the Stat Dept Computer Lab (CH-341) need to *FTP* the graphic file(s) to a computer with access to a printer and then use *Adobe's Acrobat Reader* to view and print out a hard-copy. To do so,

- Quit `S-PLUS` (type `q()` at the prompt).
- Create a pdf file by typing at the UNIX prompt:

```
ps2pdf myplot.ps myplot.pdf
```

- Use an *FTP* program on your computer (PC/Mac/UNIX) and connect to `ftp.stat.ohio-state.edu` and type in your Stat Dept user name and password. For example, the *FTP* program on the Macs is *Fetch*.

'Classical', Univariate, Tests in S-PLUS

The available univariate tests functions are:

```
binom.test      chisq.test
cor.test        fisher.test
friedman.test   kruskal.test
mantelhaen.test mcnemar.test
prop.test       t.test
var.test        wilcox.test
>
```

Example: t-test

As an example, test if the mean mileage of medium-sized cars is different than that of sporty cars, using the data in `fuel.frame`. This test can be carried out with the function `t.test`, which has the following arguments:

```
> args(t.test)
function(x, y = NULL, alternative = "two.sided",
         mu = 0, paired = F, var.equal = T,
         conf.level = 0.95)
NULL
>
```

The function can compare two samples, x and y (paired or not), or just one sample, x , to a given mean, μ .

Our example:

```
> attach(fuel.frame)
> t.test(Mileage[Type=='Medium'],
+        Mileage[Type=='Sporty'],
+        var.equal=F) ## assume unequal variance
```

```
Welch Modified Two-Sample t-Test
```

```
data: Mileage[Type == "Medium"] and
      Mileage[Type == "Sporty"]
t = -2.7968, df = 8.478, p-value = 0.022
alternative hypothesis:
true difference in means is not equal to 0
95 percent confidence interval:
 -7.6851619 -0.7763765
sample estimates:
mean of x mean of y
 21.76923      26

> detach('fuel.frame')
NULL
>
```

That is, the t test statistic is -2.80, with 8.478 degrees of freedom, resulting in a p-value of 0.022. That is, the

hypothesis $H_0 : \mu_x = \mu_y$ is rejected at α -level = 5% — as can also be seen by observing that the 95% confidence interval for $\bar{x} - \bar{y}$ does not contain zero.

Models

Linear Regression: the `lm` function

The `lm` function in S-PLUS fits *linear models* (*multiple regression*).

Unweighted Logit Analysis (ULA)

As an example, fit *ULA* model to the `tox` data (as done in class). That is, to fit the model:

$$\log \left(\frac{p_i}{100 - p_i} \right) = \alpha + \beta \log(d_i) + \epsilon_i, \quad i = 1, \dots, 8,$$

where the p_i 's are the percentage killed, $x_i = \log(d_i)$ is the log concentration, and $\epsilon_i \sim N(0, \sigma^2)$, independently. That is, ULA reduces to *simple linear regression*.

```
> ula <- lm(logit.p ~ log.d, data=tox)
>
```

The first argument is a *formula*, specifying the model to fit, and the second one is a `data.frame` where the variables can be found (optional argument).

In the formula `logit.p ~ log.d`, the response variable is to the left of `~` symbol, and explanatory variables on the right side of the symbol. We give a few examples about how to specify different models in **S-PLUS**. Let `y` be a numerical response variable, and `v` and `t` be two numerical explanatory variables; then:

- `formula = y ~ -1 + v`, is the model $y_i = \beta v_i + \epsilon_i$ (no intercept).
- `formula = y ~ v + t`, is the model $y_i = \alpha + \beta_1 v_i + \beta_2 t_i + \epsilon_i$.
- `formula = y ~ v + t + v:t`, is the model $y_i = \alpha + \beta_1 v_i + \beta_2 t_i + \beta_3 v_i t_i + \epsilon_i$

It is also possible to use *categorical* (a variable of class `factor` in **S-PLUS**) to specify formulas. For example, if `A` is a factor variable in **S-PLUS** with 3 levels, then the formula, `y ~ A + v`, is the model $y_{ik} = \alpha_k + \beta v_{ik} + \epsilon_i$ (different intercept for each level of `A`). The formula, `y ~ A + A:v`, fits a different intercept and slope for each level.

The object created by `lm` is of *class* `lm`, with functions to `print`, create a `summary`, and `plot` the resulting fit (the `lm` object):

```
> ula ## the print-out
```

```
Call:
```

```
lm(formula = logit.p ~ log.d, data = tox)
```

```
Coefficients:
```

```
(Intercept)    log.d  
  3.212505  2.166482
```

```
Degrees of freedom: 7 total; 5 residual
```

```
Residual standard error: 0.1521564
```

```
> summary(ula) ## the summary
```

```
Call: lm(formula = logit.p ~ log.d, data = tox)
```

```
Residuals:
```

```
      1      2      3      4      5      6  
0.1919 -0.1269 0.01838 -0.1341 -0.1527 0.08595
```

```
      7  
0.1175
```

```
Coefficients:
```

```
              Value Std. Error t value Pr(>|t|)  
(Intercept)  3.2125  0.1040  30.8953  0.0000  
      log.d   2.1665  0.0747  28.9896  0.0000
```

```
Residual standard error: 0.1522 on 5 degrees of freedom
Multiple R-Squared: 0.9941
F-statistic: 840.4 on 1 and 5 degrees of freedom, the p-
value is 9.153e-07
```

```
Correlation of Coefficients:
```

```
(Intercept)
```

```
log.d 0.8331
```

```
> par(mfrow=c(2,3)) ## 2 rows and 2 col of plots
```

```
> plot(ula) ## get 6 plots on one page.
```

```
>
```

The functions `coef`, `fitted` and `resid` can be used to extract the estimated coefficients, fitted values, and residuals, respectively, from the `lm` object `ula` (e.g., enter `fitted(ula)`).

To compute LC50 for this data ($LC50 = \exp(-\alpha/\beta)$)

```
> co <- coef(ula)
```

```
> co ## just to show what it does...
```

```
(Intercept)    log.d
```

```
3.212505 2.166482
```

```
> lc50 <- exp(-co[1]/co[2])
```

```
> lc50
```

```
(Intercept)
```

```
0.2269965
```

```
>
```

To plot the observed data and the fitted line (along with the LC50):

```
> par(mfrow=c(1,1)) ## one plot on device
> plot(tox$log.d, tox$logit.p,
+       xlab="log Con.",ylab="logit P")
> abline(coef(ula)) ## or lines(log.d,fitted(ula))
> abline(h=0,lty=2) ## horizontal 50% line
> abline(v=log(lc50),lty=2) ## the LC50 line
>
```

Generalized Linear Models

A more accurate Logit Analysis is to compute the MLEs for the intercept and the slope. The function `glm` can be used to fit a *generalized linear model* (GLM), and returns a object of type `glm`. For our `tox` data:

```
> la.glm <- glm(cbind(r,n-r) ~ log.d, data=tox,
+              family=binomial(link=logit))
> la.glm ## the print-out for glm object
```

Call:

```
glm(formula = cbind(r, n - r) ~ log.d, family =
     binomial(link = logit), data = tox)
```

Coefficients:

```
(Intercept)    log.d
  3.123613  2.127853
```

```
Degrees of Freedom: 7 Total; 5 Residual
Residual Deviance: 0.7336142
> summary(la.glm) ## the summary for glm object

Call: glm(formula = cbind(r, n - r) ~ log.d, family =
          binomial(link = logit), data = tox)
Deviance Residuals:
     1         2         3         4
0.4838426 -0.3603953  0.165713 -0.320634
     5         6         7
-0.2304975  0.3171778  0.2926225

Coefficients:
                Value Std. Error  t value
(Intercept)  3.123613   0.3349208  9.326424
      log.d   2.127853   0.2214073  9.610583

(Dispersion Parameter for Binomial family taken to be 1

Null Deviance: 146.0218 on 6 degrees of freedom
Residual Deviance: 0.7336142 on 5 degrees of freedom

Number of Fisher Scoring Iterations: 3

Correlation of Coefficients:
      (Intercept)
log.d  0.915217
```

>

The LC50 in this case is:

```
> co <- coef(la.glm)
> co    ## just to show what it does...
(Intercept)    log.d
   3.123613  2.127853
> lc50 <- exp(-co[1]/co[2])
> lc50
(Intercept)
   0.2303939
>
```

This is a slightly different value from that given by ULA (0.2269965).